Proceedings of the 10th WSEAS International Conference on COMPUTERS, Vouliagmeni, Athens, Greece, July 13-15, 2006 (pp1196-1201)

Enhancing Real-Time CAN Communications by the Prioritization of Urgent Messages at the Outgoing Queue

ANTÓNIO J. PIRES[(1)], JOÃO P. SOUSA[(2)], FRANCISCO VASQUES[(3)]
[1,2,3] Faculdade de Engenharia da Universidade do Porto
[2] INESC Porto
Rua Dr. Roberto Frias, 4200-465 Porto
PORTUGAL
{julio.pires, jpsousa, vasques}@fe.up.pt

Abstract: - To ensure the correct behaviour of a Networked Control System, the communication network must provide a reliable and timely communication service. The two components with the highest impact on the communication delays are the Medium Access Control (MAC) protocol and the local communication stack, therefore, the usage of an adequate communication stack is of utmost importance to guarantee the timing correctness of a feedback control application.

In this paper, we propose the use of state-of-the-art scheduling algorithms to manage the outgoing queue of a local communication stack. We demonstrate that it is possible to improve the responsiveness of applications supported by the CAN communication protocol, by using just a light scheduling middleware to adequately schedule the outgoing queue. We also show that implementing such middleware even on top of COTS communication hardware, opens the possibility to enhance the communication process by minimizing the number of deadline misses for highly loaded network scenarios.

Keywords: - CAN Communication, Fieldbus Systems, Real-Time Communication, Networked Control Systems, Priority Inversion, Message Scheduling.

1 Introduction

Fieldbus networks became increasingly popular in computer-controlled systems. Computer-controlled systems wherein the control loops are closed through a real-time fieldbus network are called Networked Control Systems (NCS). Figure 1 represents an example of a Networked Control System.

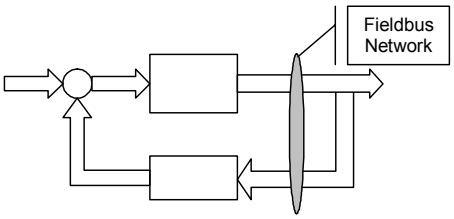

Figure 1: Networked Control System (NCS)

A fieldbus network interconnecting sensors, actuators and controllers in a feedback control system, must be able to:
a) support periodic message streams, in order to convey the control-related periodic data between the controller and the set of related sensors / actuators;
b) guarantee upper-bounded response times for the message transfers, in order to cope with the control-related delays;
c) guarantee a predictable timing behavior in the presence of a variable network load due to traffic non related to the control application (such as alarms, surveillance video/audio streams, etc.);
d) guarantee a predictable timing behaviour in the presence of a faulty communication behaviour.

In addition, a well-known problem when using a fieldbus network is the presence of induced jitter, that is, the variability of the time interval between consecutive transfers. For instance, in spite of periodically requesting the transfer of a specific sensor value, the actual transfer will not be immediately executed, as messages need to be scheduled for transmission in a shared resource (the communication medium). As a consequence, in some of the control cycles the sensor message will be transferred earlier in the cycle period, and in some other cycles it will be transferred later. The real-time service provided by the fieldbus network will just guarantee that the sensor message will always be transferred before its deadline.

Controller Area Network (CAN) [1] was originally designed for use within road vehicles, to solve cabling problems arising from the growing use of microprocessor-based components in vehicles. Due to its interesting characteristics, CAN is also being considered for the automated manufacturing and distributed process control environments [2], to support small-scale Networked Control Systems.

Several studies on how to guarantee the timing requirements of messages in CAN networks are available (e.g. [3]), thus providing pre-run-time schedulability conditions for the analysis of the timing requirements of NCS traffic. More recently, the CAN communication protocol has attracted an increased attention from the research community, namely in what concerns the guarantee of real-time constraints [4], the implementation of innovative scheduling approaches, either server-based [5] or cyclic table-based [6], and the provision of reliable communication services [7].

However, when assessing the most common COTS communication boards, one of the perceived drawbacks is that the outgoing communication queues are FIFO queues. Therefore, the message transfer in CAN networks using such communication boards is prone to priority inversions. That is, lower priority messages will be regularly transferred before higher priority messages. As a consequence, message deadlines will be frequently missed, whereas they could be respected if the adequate scheduling strategies were implemented.

This paper assesses a set of state-of-the-art scheduling algorithms that have been proposed for scheduling messages in CAN networks. Specifically, it assesses how such algorithms can be implemented to re-order the outgoing communication queue and, as a consequence, to guarantee the real-time behavior of CAN communications, minimizing or even avoiding the priority inversion problem. The main goal of this paper is to demonstrate that, using COTS hardware and a light scheduling software layer implementing adequate real-time message scheduling strategies, it is possible to improve the responsiveness of real-time applications supported by the CAN communication protocol.

The remainder of this paper is organized as follows. Section 2 describes some of the most relevant available research works on response time analysis of CAN networks. In Section 3 we briefly describe the experimental setup used to implement and to assess the proposed middleware. Then, in Section 4, we present the main strategies to schedule the local outgoing queues, which are proposed in this paper. These strategies will be assessed in Section 5. Finally, some conclusions are drawn in Section 6.

2 CAN Real-Time Communications

In [3], the authors addressed in detail the analysis of real-time communications in CAN, assuming fixed priorities for message streams. In such case, the worst-case response time of a queued message, measured from the release of the queuing task to the time the message is fully transmitted, is:

$$R_m = J_m + I_m + C_m \tag{1}$$

J_m is the queuing jitter of message stream S_m, inherited from the worst-case response time $R_{sender(m)}$ (where sender(m) denotes the task which queues the message m). The term I_m represents the worst-case queuing delay - longest time between placing the message in the priority-ordered outgoing queue - and the start of the message transmission.

The deadline monotonic (DM) priority assignment [8] can be directly implemented in a CAN network, by setting the identifier field of each message stream to a unique priority, according to the DM rule. Therefore:

$$I_m = B_m + \sum_{\forall j \in hp(m)} \left(\left\lceil \frac{I_m + J_j + \tau_{bit}}{T_j} \right\rceil \times C_j \right) \tag{2}$$

where B_m is the worst-case blocking factor, which is equal to the longest time taken to transmit a lower priority message, and is given by:

$$B_m = \max_{\forall k \in lp(m)} \{0, C_k\} \tag{3}$$

The set *lp(m)* is the set of message streams with lower-priority than message stream S_m. τ_{bit} is the time taken to transmit a bit on the bus and *hp(m)* is the set of message streams in the system with higher-priority than the message stream S_m.

C_m is the longest time taken to transmit a message from stream S_m. CAN has a 47 bit overhead per message, and a stuff width of 5 bits. Only 34 of the 47 bits of overhead are subject to stuffing, so C_m can be defined as:

$$C_m = \left(\left\lfloor \frac{34 + 8 \times db_m}{5} \right\rfloor + 47 + 8 \times s_m \right) \times \tau_{bit} \tag{4}$$

where db_m is the number of data bytes in the message.

Alternatives for the fixed priority assignment are the dynamic priority schemes, such as the non-preemptive Earliest Deadline First (EDF). In [2], the authors analyze how the EDF scheduling algorithm [9] could be used to schedule CAN messages. In this work, the authors propose the use of a mixed traffic scheduler (MTS), which attempts to provide a high utilization of the communication medium while using the standard 11-bit format for the identifier field.

The goal of the MTS scheduler is to make the identifier fields of different message streams to reflect the deadlines of messages. However, considering that each message must have a unique identifier field (which is a requirement of CAN), they suggested the division of the identifier field into three

sub-fields, as shown in Figure 3, where a) is for the messages that are to be scheduled according to the EDF. b) is for messages to be scheduled according to the DM, and c) is for low-priority messages.

For the higher-priority message, the deadline field is derived from the deadline of the message. To deal with the case where two messages have the same deadline, the one with the highest uniqueness code will win (note that Zuberi and Shin assume a Wired-OR bus, thus being '1' the dominant bit). The uniqueness code also serves to identify the message for reception purposes.

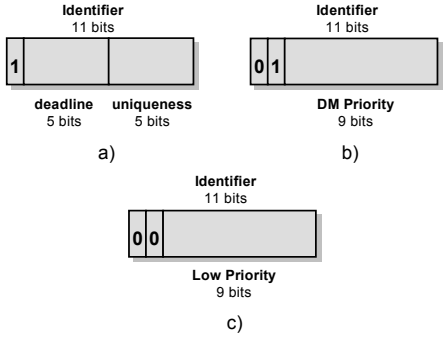

Figure 3: Identifier field of a MTS message

This type of encoding raises two problems: the first is that the remaining slack time of a message changes with every clock tick. This requires identifiers of all messages to be continually updated, and also that each local clock must be synchronized. The second is that in a typical system, message streams may have largely different deadlines, which raises a problem with the length of the identifier field (only 5 bits to encode the deadline).

To solve the second problem, the authors divided the time into regions and encoded deadlines according to which region they fall in. Deadlines are then expressed relatively to a periodically increasing reference called the start of epoch (SOE). Figure 4 illustrates this concept for m = 2.

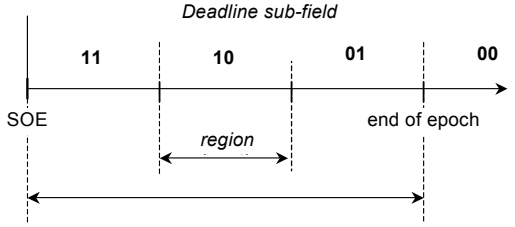

Figure 4: Deadline field, encoded in only two bits

3 Experimental Setup

In order to define an adequate experimental setup, we have made a careful analysis of the available COTS components for CAN communications, both for hardware and software.

3.1 Hw Framework

When selecting a hardware framework, important topics are the microcontroller family, the memory size (both program and data memory), the internal peripherals of the microcontroller and the available programming tools (and its cost).

After the analysis of all these characteristics, we have set up a framework with the following items:

1) One IXXAT Automation iPC-I320 board in a desktop PC, for the reception/transmission of control data in the CAN.
2) Several Phytec PhyCORE AduC812 modules for the remote data acquisition or data output nodes. (Figure 5).

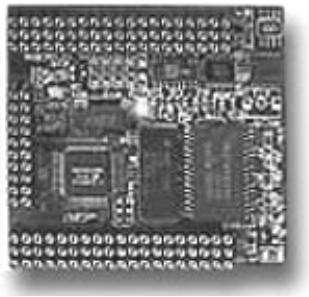

Figure 5: PHYTEC phyCORE AduC812

The main purpose of this framework was to implement and assess a set of state-of-the-art scheduling algorithms that have been proposed for scheduling messages in CAN networks. Such scheduling strategies were implemented in the PhyCORE modules, and a desktop PC, through the IXXAT Automation iPC-I320 board, was used to monitor the timing characteristics of the network traffic.

3.2 Network Traffic Load

A highly loaded network scenario was implemented, in order to assess the proposed scheduling strategies. Specifically, we are interested in the comparison of the timing results (number of deadline misses) when different message scheduling approaches are used to handle a highly loaded network scenario.

Therefore, the first step is to derive such load scenario. We consider a CAN network transferring 8 message streams, with priorities set according to the Deadline Monotonic rule [8], that is: the message stream with the smaller deadline (d_i) will be assigned with the highest priority identifier. All the transferred messages have the same message length: C_i=5,5ms and a deadline value equal to its periodicity ($d_i = T_i$).

The periodicities of the message streams were set according to the following rules:

a) The message stream with smaller periodicity generates a message every 30ms;
b) The periodicity (Ti) of the other message streams are related by a integer coefficient a, which enables setting up different network loads for related sets of message streams.

Therefore, the overall network load (U) can be evaluated using the following equation:

$$U = 2 \times \left(\frac{C}{T_1} + \frac{C}{2aT_1} + \frac{C}{4aT_1} + \frac{C}{6aT_1} \right) \quad (6)$$

Considering that T_1 = 30ms and C=5,5ms, the a parameter can be expressed as follows:

$$a = \frac{121}{360U - 132} \quad (7)$$

For the experimental setup, the eight above defined message stream sets were used, with a network load ranging from 70% up to 95%. Figure 6 illustrates two of those message stream sets, taken from an experience with only two remote nodes (odd and even streams).

Message Stream	C_i (ms)	T_i, d_i (ms)	U_i (%)
F1	5,5	30	18,3
F2	5,5	30	18,3
F3	5,5	60	9,2
F4	5,5	60	9,2
F5	5,5	121	4,5
F6	5,5	121	4,5
F7	5,5	181	3,0
F8	5,5	181	3,0
		total:	70,2%

	C_i (ms)	T_i, d_i (ms)	U_i (%)
F1	5,5	30	18,3
F2	5,5	30	18,3
F3	5,5	35	15,7
F4	5,5	35	15,7
F5	5,5	69	8,0
F6	5,5	69	8,0
F7	5,5	104	5,3
F8	5,5	104	5,3
		total:	94,6%

Figure 6: Example of 2 message stream sets

Finally, the message scheduling was observed during a *60s* snapshot and the number of deadline misses for each scheduling strategy were counted and compared. The total number of generated messages during the *60s* snapshot, for each message stream and for each network load, can be observed in Figure 7.

Message	Network Load					
	70%	75%	80%	85%	90%	95%
F1	2000	2000	2000	2000	2000	2000
F2	2000	2000	2000	2000	2000	2000
F3	1000	1132	1276	1428	1579	1714
F4	1000	1132	1276	1428	1579	1714
F5	496	572	645	723	790	870
F6	496	572	645	723	790	870
F7	332	380	429	480	531	577
F8	332	380	429	480	531	577

Figure 7: Total number of generated messages

4 Scheduling of the Outgoing Communication Queue

Our main goal was to assess a set of state-of-the-art scheduling algorithms, used to schedule messages in CAN networks. Specifically, we were interested in the assessment of the outgoing communication queue and, as a consequence, to guarantee the real-time behaviour of the CAN communication protocol.

Three scheduling strategies were devised to manage the software-implemented outgoing communication queue. The implemented setup is briefly described in Figure 8.

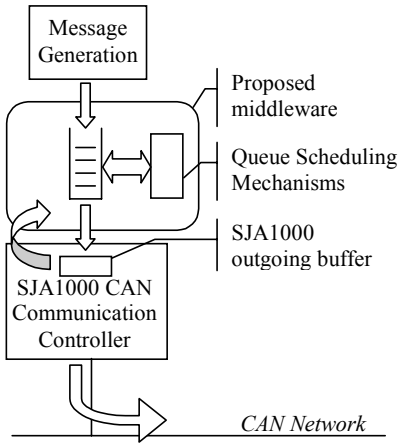

Figure 8: Inside a remote node

4.1 Message generation

Messages are generated at each node by the periodic interrupt of one of the microcontroller's internal timers and according to several predefined periodicities. A maximum of four message streams are generated at each node and stored on the outgoing message queue according to one of three possible scheduling disciplines.

4.2 Local FIFO Scheduling

This scheduling strategy is implemented just for comparison purposes. This is the algorithm

traditionally used in CAN communication boards for scheduling the outgoing message queue. In order to enable a direct comparison between the results obtained when using different scheduling strategies, we implemented the FIFO Scheduling algorithm in the proposed middleware using a circular buffer with monotonic input and output pointers.

4.3 Local Priority Scheduling

This scheduling strategy manipulates the positions where generated messages are placed. Such manipulation is supported by a timed pointer incremented every 5ms. Whenever a message is generated and moved to the circular buffer, it is placed in the (*pointer* + T_i) position, where T_i is the periodicity of message stream M_i expressed in multiples of 5 milliseconds.

This means that the message with the earliest deadline in the circular buffer is the one that will be scheduled for transmission.

If the SJA1000 is unable to immediately transfer the scheduled message and a newly arrived message to the circular buffer has an earlier deadline, then those two messages will be swapped. This means that the previously scheduled message is moved back to the circular buffer.

4.4 Local Priority Scheduling with Temporal Epochs

The third scheduling strategy implements the concept of Temporal Epochs proposed by Zuberi and Shin [2], that was briefly explained in Section 2. The proposed scheme implements the Mixed Traffic Scheduler (MTS) to schedule messages similarly to the EDF scheduling algorithm [9]. Basically, it encodes the deadline field (*DF*) according to the following set of rules (as detailed in Figure 3a):

$$DF = \begin{cases} \dfrac{x - timeout}{16}, \text{if } \dfrac{x - timeout}{16} < 15 \\ 255 \quad , \text{if } \dfrac{x - timeout}{16} \geq 15 \end{cases} \quad (8)$$

together with the implementation of the Local Priority Scheduling proposed in the previous subsection.

5 Experimental Evaluation

The assessment of the proposed scheduling strategies was done for a highly loaded network, according to the Network Traffic Load scenario defined in subsection 3.2.

5.1 Deadline misses

The obtained results during a *60s* snapshot are illustrated in Figures 9, 10 and 11.

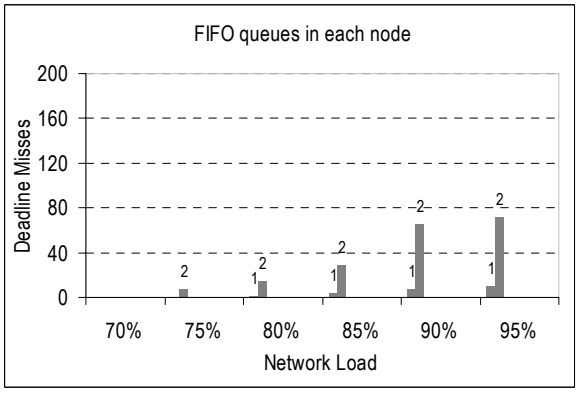

Figure 9: Results for FIFO Scheduling

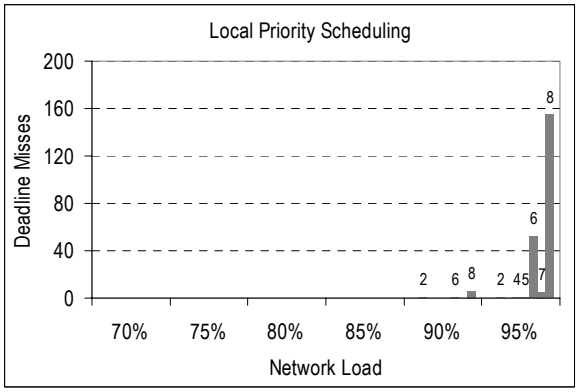

Figure 10: Results for Local Priority Scheduling

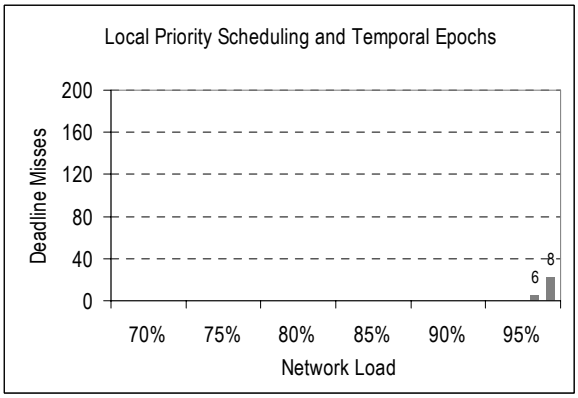

Figure 11: Results for Local Priority Scheduling with Temporal Epochs

Clearly, the number of deadline misses is totally unacceptable for the case of the FIFO Scheduling approach as there are deadline misses for the highest priority streams M1 and M2 even with network loads as small as 75%.

With the Local Priority Scheduling strategy, the results are significantly different. While for intermediate network load scenarios, there were no experienced deadline misses, for higher network load scenarios there was a significant increase in the number of deadline misses, but now just for the lowest priority streams M6 and M8.

Finally, when implementing the Local Priority Scheduling with Temporal Epochs, the obtained results are clearly better than the results for any of the previous scheduling strategies. The number of deadline misses both for intermediate and for highly loaded network scenarios is significantly smaller than in the previous cases. This behaviour forecasts an improved capability to support real-time applications.

5.2 Scheduling code overhead

The three implemented strategies were scrutinized for code overhead and the results of this analysis are illustrated in Figure 12.

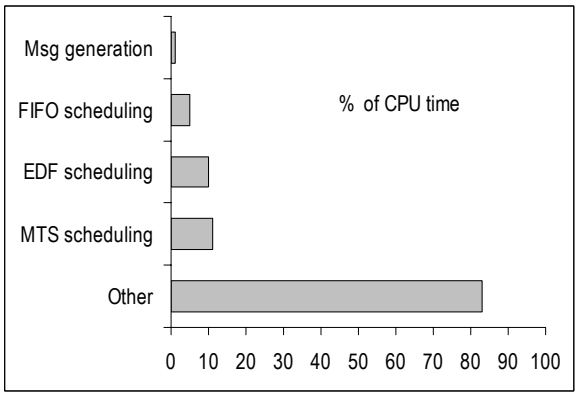

Figure 12: Code profiler results

As it can be observed, the results of the code profiler tool reveal that the middleware implementation represents an overhead smaller than 12%, even for the case of the used small performance microcontroller. When comparing the proposed scheduling approaches, it also reveals a very small code overhead for the second and third strategies over the FIFO strategy.

6 Conclusions

In this paper, we propose the use of state-of-the-art scheduling algorithms to manage the outgoing queue of CAN communication stacks, in order to improve the responsiveness of supported applications. We showed that it is possible to reduce the occurrence of priority inversions in the communication medium, and therefore it becomes possible to decrease the number of deadline misses even for highly loaded network scenarios.

We provided early implementation results of deadline misses versus network load for two scheduling algorithms and showed their better performance and small computing overhead when compared with the FIFO scheduling method, making them very attractive for implementation on low end processors.

References:

[1] ISO 11898. (1993). “Road Vehicle - Interchange of Digital Information - Controller Area Network (CAN) for High-Speed Communication”. ISO.

[2] Zuberi, K. and Shin, K. (1997). “Scheduling messages on Controller Area Network for Real Time CIM Applications”. In IEEE Transactions on Robotics and Automation, Vol. 13, No. 2, pp 310-314.

[3] Tindell, K., Burns, A. and Wellings, A. (1995). “Calculating Controller Area Network (CAN) Message Response Time”. In Control Engineering Practice, Vol. 3, No. 8, pp. 1163-1169.

[4] Cavalieri, S., “Meeting Real-Time Constraints in CAN”, IEEE Trans. on Industrial Informatics, 2005, vol. 1, no. 2, pp. 124-135.

[5] Nolte, T.; Nolin, M.; Hansson, H.A., “Real-Time Server-Based Communication With CAN”, IEEE Trans. on Industrial Informatics, 2005, vol. 1, no. 3, pp. 192-201.

[6] L. Almeida, P. Pedreiras, J. Fonseca, “The FTT-CAN protocol: why and how”, *IEEE Transactions on Industrial Electronics*, Volume 49, Issue 6, Dec. 2002 Page(s):1189 - 1201

[7] L. Pinho, F. Vasques, “Reliable Real-Time Communication in CAN Networks", IEEE Transactions on Computers, Volume 52, Issue 12, Dec. 2003 Page(s): 1594 - 1607.

[8] N. Audsley A. Burns M. Richardson A. Wellings, “Hard Real-Time Scheduling: The Deadline Monotonic Approach”. In *Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software*, Atlanta, 1991.

[9] C. Liu, J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”. In *Journal of the ACM*, Vol 20, No. 1, January 1973, pp. 46-61.

[10] K. Etschberger; *Controller Area Network: Basics, Protocols, Chips and Applications*; IXXAT Automation, 2001.